GESTIC INCONTRA ARDUINO

Con la board qui descritta, interfacciamo la scheda elettrodi per il riconoscimento dei gesti ad Arduino e Raspberry Pi. Vediamo per prima l'applicazione con Arduino.

di MATTEO DESTRO

Nella precedente puntata abbiamo descritto nei dettagli le elettroniche degli elettrodi e della scheda demo che abbiamo progettato per gestire gli stessi, nonché la scheda di interfaccia tra gli elettrodi e il PC da usare per la parametrizzazione. In questa puntata parleremo del nostro software da interfacciare alla scheda demo e del nuovo elettrodo sviluppato per interfacciarsi ad Arduino Uno Rev. 3 e Raspberry Pi. Per studiare la piattaforma Gestic abbiamo pensato di realizzare un software con cui interfacciarsi alla nostra scheda demo, grazie al quale possiamo vedere in tempo reale le gesture riconosciute dall'integrato MGC3130. Il software GesticTester, come mostrato dalla **Fig. 1,** è suddiviso in due TAB distinti: il primo (Gesture) monitorizza e visualizza tutte le gesture intercettate dall'integrato MGC3130, mentre il secondo (Firmware) serve per visualizzare la revisione firmware caricata nell'integrato MGC3130. La comunicazione tra PC e scheda demo avviene tramite interfaccia USB; per instaurare una comunicazione si deve prima di tutto selezionare dal menù "***Communication***" la voce "***Open communication***" la quale richiama la finestra di selezione della COM virtuale cui è connessa la scheda demo. La **Fig. 2** mostra la finestra di selezione della COM virtuale. Notate che la COM può essere selezionata manualmente, oppure, più

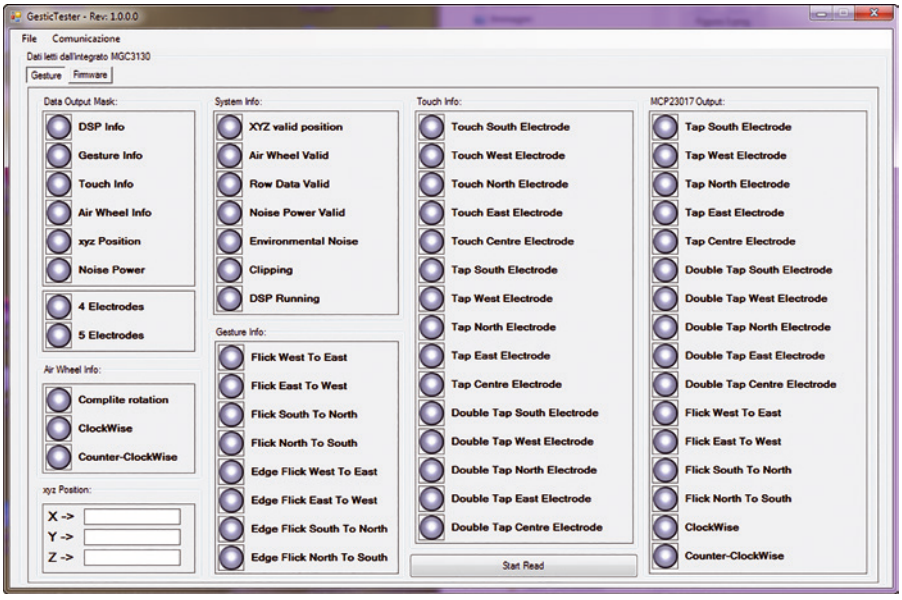


Fig. 1

comodamente, con la ricerca automatica della scheda che si avvia cliccando sulla voce "***Auto Find Device***". Sulla destra compare una "***Text box***" dove vengono listati i comandi che il driver di comunicazione invia alla scheda, mentre in basso, se attiva, c'è una seconda "***Text box***" in cui sono listati i byte dei dati che passano sulla COM virtuale (in pratica sono i pacchetti dati che dal PC vengono inviati alla scheda demo e le relative risposte). Per attivare la seconda "***Text box***" si deve cliccare su "***Start Logger***".

Se la comunicazione viene instaurata con successo, comparirà il nome del device collegato (in questo caso "***Gestic DemoBoard***") e il relativo FW caricato in esso.

Instaurata la comunicazione con la scheda demo, si può iniziare a monitorare le gesture riconosciute dall'integrato MGC3130. Per attivare la scansione, cliccare sulla voce "***Start Read***" (**Fig. 1**).

La sezione "***Gesture Info***" evidenzia le gesture di movimento della mano ovvero:

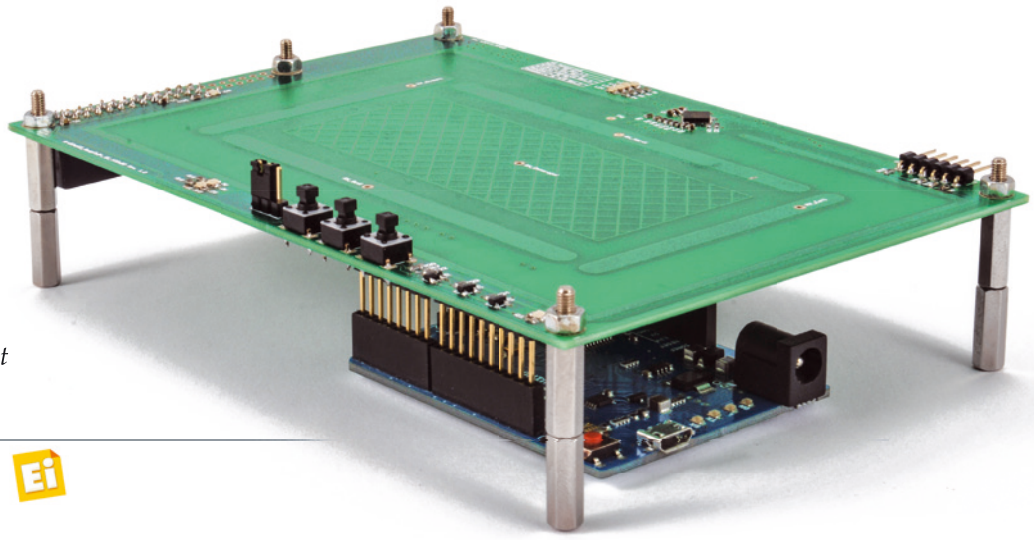
- *movimento da Ovest a Est, compreso movimento Ovest – Est sul bordo (Edge Flick);*
- *Movimento da Est a Ovest, compreso movimento Est – Ovest sul bordo (Edge Flick)*;
- *Movimento da Sud a Nord, compreso movimento Sud – Nord sul bordo (Edge Flick)*;
- *Movimento da Nord a Sud, compreso movimento Nord – Sud sul bordo (Edge Flick).*

La sezione "***Touch Info***" evidenzia le gesture di tocco sugli elettrodi; in questo caso si identificano le gesture "***Touch***", "***Tap***" e "***Double Tap***" per un totale di quindici diverse gesture. Vi ricordiamo che con il termine Tap si intende toccare l'elettrodo con un colpetto veloce, che è diverso dalla gesture Touch, la quale è un movimento più lento. Infine la sezione "***MCP23017 Output***" la quale evidenzia le gesture riportate sulle uscite digitali gestite dall'integrato MCP23017.

UN NUOVO ELETTRODO, ARDUINO E RASPBERRY PI

Per sfruttare le potenzialità dell'integrato MGC3130 abbiamo pensato di sviluppare un nuovo elettrodo con la possibilità di collegare, oltre che la nostra scheda demo vista nella precedente puntata, anche la scheda Arduino Uno Rev3 (oppure la Arduino Leonardo Rev3). Inoltre il nuovo elettrodo è stato pensato e progettato per poter collegare anche le schede Raspberry Pi e in particolare la Raspberry Pi B+ o Raspberry Pi 2. Ovviamente solo una delle schede sopra citate può essere direttamente collegata alla scheda elettrodo; in altre parole, se decidiamo di lavorare con il mondo Arduino non possiamo collegare anche la Raspberry Pi e viceversa. Resta sottinteso che il discorso di unicità di connessione verso l'elettrodo vale anche per la nostra scheda. In questo articolo descriveremo l'accoppiata con Arduino, rimandando al prossimo quella con la Raspberry Pi.

Iniziamo con il descrivere lo schema elettrico del nostro nuovo elettrodo partendo dall'integrato MGC3130, il quale è configurato per gestire cinque



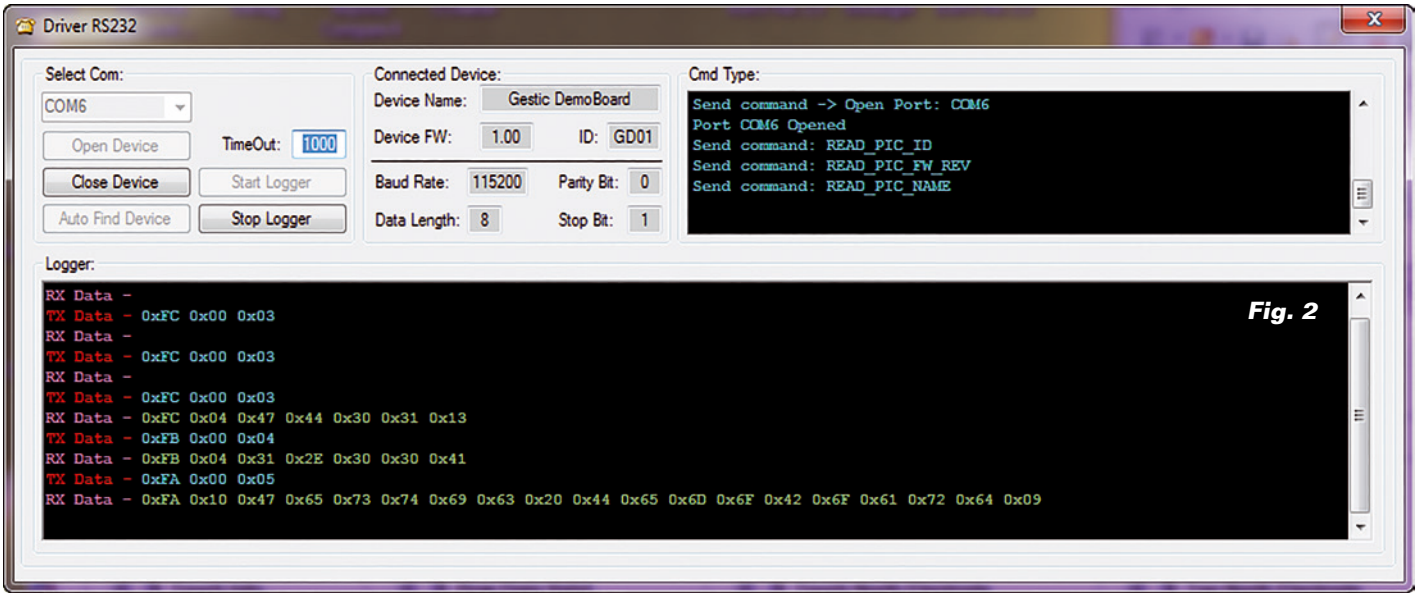


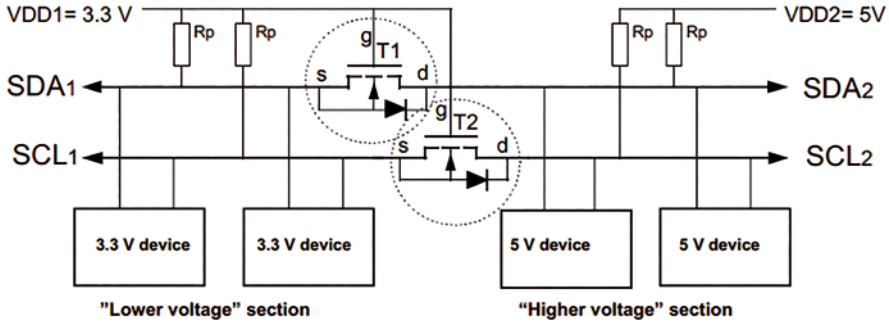
Fig. 2

elettrodi di ricezione (R_X) e un elettrodo di trasmissione (T_X). Per ognuno degli elettrodi di ricezione è stata predisposta una resistenza da 10 kΩ, la quale, grazie alla capacità di ingresso dei pin R_{X0}, R_{X1}, R_{X2}, R_{X3} e R_{X4} crea un filtro passa basso eliminando le interferenze ad alta frequenza. Anche per l'elettrodo T_X è stata predisposta una resistenza, che attualmente è un ponticello (0 Ω). L'interfaccia di comunicazione utilizzata dall'integrato MGC3130 è il consueto bus I²C, più le due linee di comunicazione TS (EIO0) e RESET, il quale viene portato sia all'interfaccia di connessione verso le schede Arduino (U2) che all'interfaccia di connessione della scheda Raspberry Pi (U3). Tali linee vengono ovviamente portate anche ai connettori CN1 e CN2 per la connessione della nostra scheda demo. Ricordiamo che la scheda è collegabile sia utilizzando un cavo Wurth da 10 poli (passo 1mm, lunghezza 50 mm) al connettore CN2 oppure direttamente tramite connettore CN1. Dato che l'integrato MGC3130 è alimentato a +3,3 Vcc si rende necessario adattare i segnali verso l'elettronica delle schede Arduino in quanto alimentate a +5 Vcc. L'adattamento di linea viene fatto sfruttando i MOSFET BSS123 a canale N (Q1, Q2 e Q3). Vediamo in pratica come funziona l'adattamento di linea commentando la **Fig. 3**. Un traslatore di livello bidirezionale è utilizzato per interconnettere due sezioni di un BUS I²C (vale anche per SPI o qualsiasi linea bidirezionale) dove ogni sezione è alimentata con un livello di tensione differente, nel nostro caso +3,3 Vcc per l'elettronica dell'elettrodo e +5 Vcc per quanto riguarda le schede Arduino.

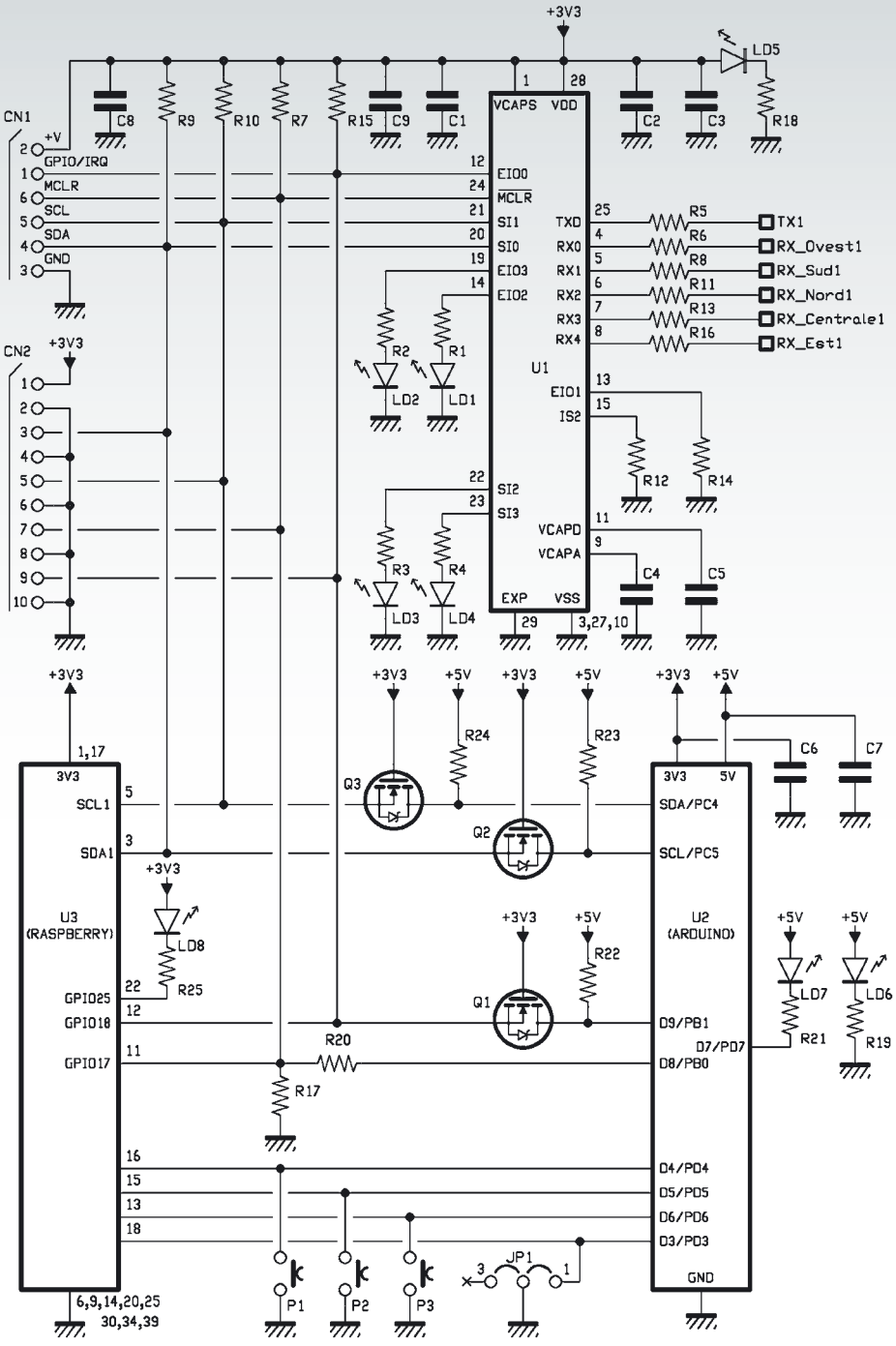
La sezione a bassa tensione, a sinistra, prevede due resistenze di pull-up collegate al source del MOSFET, per le linee SDA e SCL, mentre il gate viene collegato direttamente alla tensione di alimentazione più bassa, nel nostro caso +3,3 V. La sezione ad alta tensione prevede anch'essa due resistenze di pull-up collegate al drain del MOSFET. Per entrambe le sezioni le periferiche devono essere configurate come uscite open-drain. Il MOSFET utilizzato per entrambe le linee è un MOSFET ad accrescimento canale N con il substrato internamente connesso al source. Inoltre si devono scegliere MOSFET con già integrato il diodo che connette il substrato con il drain creando una giunzione n-p. I MOSFET BSS123 da noi scelti soddisfano queste condizioni.

Vediamo ora in dettaglio i tre stati di funzionamento (il ragionamento lo applichiamo a

Fig. 3



[schema **ELETTRICO**]



una sola linea ma ovviamente vale per entrambe).

1. Nessuna periferica impone un livello di tensione basso sulla linea. Quindi sul lato bassa tensione la resistenza di pull-up impone un livello logico alto, di conseguenza la V_{GS} non supera la soglia del MOSFET, in quanto V_G e V_S hanno la medesima tensione, quindi il MOSFET non entra in conduzione; di conseguenza sul lato ad alta tensione la linea si ritrova a un livello logico alto imposto dalla rispettiva resistenza di pull-up.
2. La periferica a bassa tensione impone sulla linea un livello di tensione basso di conseguenza il source del MOSFET va a livello logico basso. A questo punto la V_{GS} supera la soglia e manda in conduzione il MOSFET; con il MOSFET in conduzione, il livello logico basso viene imposto anche alla sezione ad alta tensione.
3. La periferica ad alta tensione impone sulla linea un livello di tensione basso; sulla linea a bassa tensione abbiamo la resistenza di pull-up che impone un livello logico alto e quindi manda in conduzione il diodo presente nel MOSFET. La V_S scende e di conseguenza la V_{GS} supera la soglia mandando in conduzione il MOSFET; a questo punto le due sezioni hanno il medesimo livello di tensione sulla linea.

Il concetto appena esposto è stato applicato alle due linee I²C, SDA e SCL, e alla linea TS. Per quanto riguarda la linea RESET, è sufficiente la R20 assieme alla resistenza di pull-up R7. La R17 non deve essere montata.
Le linee del BUS I²C che vengono portate alla nostra scheda demo e alla scheda Raspberry Pi non serve che siano adattate, in quanto le elettroniche lavorano a +3,3 V e sono compatibili con l'integrato MGC3130.
Sia alla scheda Arduino che alla Raspberry Pi vengono collegati tre pulsanti (P1, P2 e P3) e un jumper; questi servono per riprodurre le funzioni implementate nella nostra scheda demo.
Non sono state predisposte le resistenze di pull-up perché sono già integrate sia nella scheda Arduino che nella scheda Raspberry Pi, infatti

si possono attivare da codice quando si configurano i pin del microcontrollore come ingressi. Sia per la scheda Arduino che per la scheda Raspberry Pi abbiamo predisposto un LED di segnalazione, LD7 per Arduino e LD8 per Raspberry Pi, utile durante il riconoscimento delle Gesture o durante la gestione della pressione dei pulsanti P1, P2 e P3.
Con questa nuova scheda abbiamo anche colto l'occasione per sfruttare gli I/O estesi EIO2, EIO3, EIO6 e EIO7 ai quali abbiamo collegato dei LED per segnalare le gesture riconosciute. Durante la parametrizzazione è quindi possibile decidere quale gesture deve essere monitorata e riportata su una delle possibili uscite o una combinazione di esse. Per la nostra applicazione abbiamo scelto di riportare sui LED le seguenti gesture:

- *Flick West – East;*
- *Flick East – West;*
- *Flick North – South;*
- *Flick South – North;*
- *Single Tap North;*
- *Single Tap South;*
- *Single Tap West;*
- *Single Tap East;*
- *Single Tap Centre;*
- *Clock Wise;*
- *Counter Clock wise.*

La **Fig. 4** evidenzia la configurazione delle uscite per ogni gesture di cui sopra. Ad ogni gesture riconosciuta l'integrato MGC3130 genera un impulso sulla rispettiva uscita. Tuttavia vogliamo fare notare che è possibile scegliere e configurare diversamente il comportamento delle uscite rispetto alla gesture riconosciuta. Infatti oltre all'impulso è possibile scegliere:

- *uscita permanentemente alta;*
- *uscita permanentemente bassa;*
- *toggle.*

MCP23017_Library

Oltre alla libreria scritta per la gestione dell'integrato MGC3130 è stata pensata e realizzata una libreria dedicata per la gestione dell'integrato MCP23017, che è un I/O expander della Microchip su bus I²C. Questo componente è lo stesso montato sulle schede d'espansione a relé FT1079K descritte in queste pagine.
La libreria è composta da tre file distinti "**MCP23017.c**", "**MCP23017.h**" e "**keyword.txt**". La libreria mette a disposizione una serie di funzioni utili alla gestione dell'integrato MCP23017 tra cui:

```
void    begin(void);
void    ToggleSingleBit(uint8_t RegAdd, uint8_t Bit, uint8_t i2caddr);
void    SetSingleBit(uint8_t RegAdd, uint8_t Bit, uint8_t i2caddr);
void    ResetSingleBit(uint8_t RegAdd, uint8_t Bit, uint8_t i2caddr);
void    WriteSingleReg(uint8_t RegAdd, uint8_t RegData, uint8_t i2caddr);
void    ClearReg(uint8_t RegAdd, uint8_t i2caddr);
uint8_t ReadSingleReg(uint8_t RegAdd, uint8_t i2caddr);
```

La funzione "***Begin***" serve per inizializzare la periferica I²C, la funzione "***ToggleSingleBit***" permette di eseguire il toggle su un singolo bit del registro desiderato. Infatti i parametri da passare alla funzione sono l'indirizzo del registro su cui eseguire il toggle, il bit da modificare e infine l'indirizzo hardware dell'integrato. La funzione "***SetSingleBit***" setta un singolo bit del registro desiderato, i parametri da passare sono gli stessi della precedente funzione. Viceversa la funzione "***ResetSingleBit***" che come suggerisce il nome resetta un singolo bit.
Diversamente le funzioni "***WriteSingleReg***" e "***ClearReg***" le quali rispettivamente scrivono un byte sul registro desiderato oppure resettano il registro desiderato. I parametri da passare sono i soliti: indirizzo del registro, dato che si vuole scrivere e indirizzo hardware dell'integrato.
Per ultimo "***ReadSingleReg***" la quale permette di leggere un registro dell'integrato. La funzione ritorna il byte letto, i parametri da passare sono l'indirizzo del registro da leggere e l'indirizzo hardware dell'integrato.
Oltre alle funzioni appena descritte ce ne sono altre necessarie a configurare l'integrato per il suo corretto funzionamento. A seconda della configurazione dei banchi, vedere datasheet, la sequenza di configurazione cambia. Supponendo di ragionare sul BANK0 le funzioni di inizializzazione sono le seguenti:

```
void SetAllRegBank0(uint8_t i2caddr);
void SetIntPortABank0(uint8_t Iocon, uint8_t Intcon, uint8_t Defval,
uint8_t GpintEn, uint8_t i2caddr);
void SetIntPortBBank0(uint8_t Iocon, uint8_t Intcon, uint8_t Defval,
uint8_t GpintEn, uint8_t i2caddr);
```

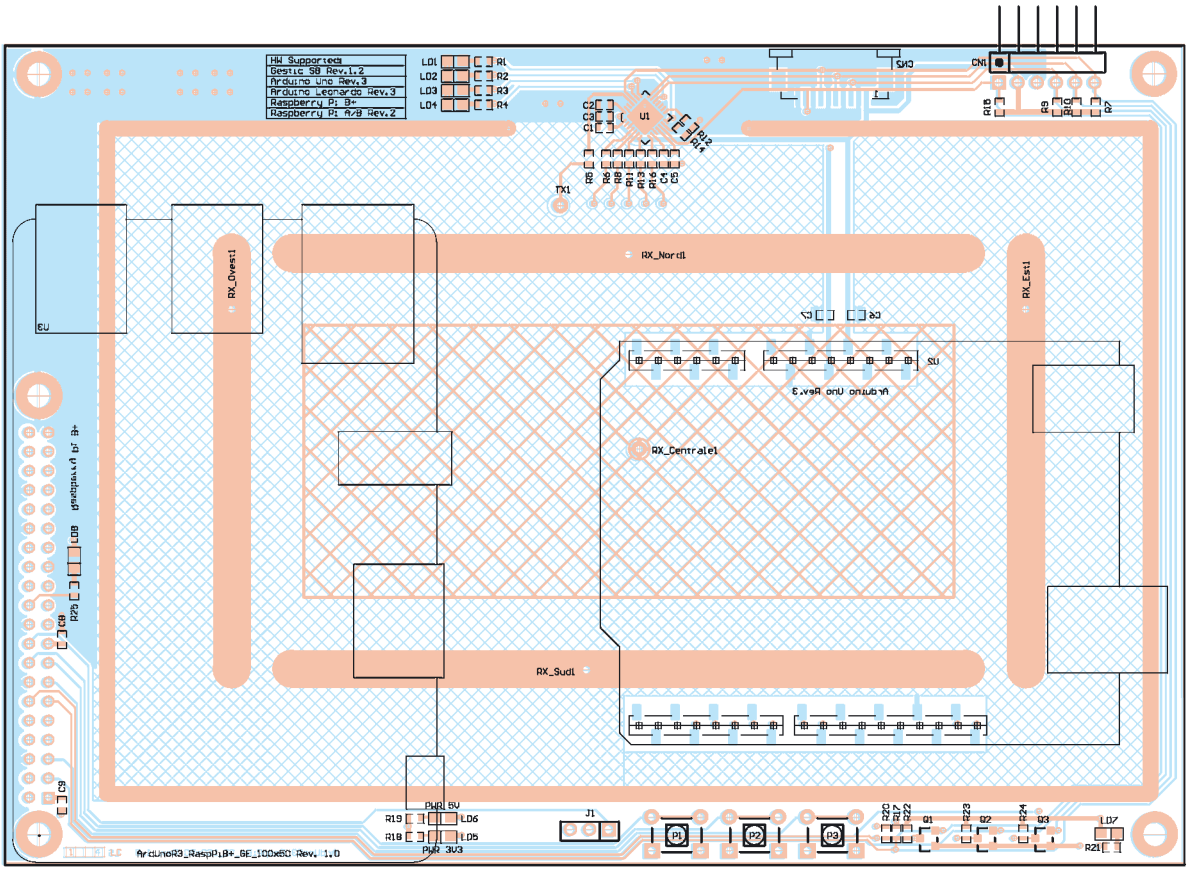
La prima funzione setta i registri dell'integrato; il parametro da passare è l'indirizzo hardware dell'integrato. Seguono poi due funzioni per configurare l'interrupt sia per la porta A che per la porta B, a patto che siano configurate come ingressi. I byte di configurazione da passare a ogni singolo registro si trovano nel file .h (MCP23017.h), l'utente può configurarli a seconda delle sue necessità.
Sarà poi la funzione "***SetAllRegBank0***" a passare ogni singolo valore ai registri da configurare. Solitamente si usa sempre la configurazione dei registri come BANK0. Nel nostro sketch, di cui parleremo a breve, abbiamo fatto uso di questa libreria, durante la fase di configurazione dei due integrati (due schede FT1079K) abbiamo usato le seguenti:

```
mcp23017.Begin();
mcp23017.SetAllRegBank0(0x00);
mcp23017.ClearReg(MCP23017_BNK0_OLATA, 0x00);
mcp23017.SetAllRegBank0(0x01);
mcp23017.ClearReg(MCP23017_BNK0_OLATA, 0x01);
```

Mentre per gestire le uscite abbiamo usato le seguenti funzioni:

```
mcp23017.ToggleSingleBit(MCP23017_BNK0_OLATA, 0, 0x00);
mcp23017.SetSingleBit(MCP23017_BNK0_OLATA, 0, 0x00);
mcp23017.ResetSingleBit(MCP23017_BNK0_OLATA, 0, 0x00);
```

Per chi volesse cimentarsi con la nostra libreria per studiarla e usarla nei propri sketch può contare su quattro sketch di esempio distribuiti assieme alla libreria.

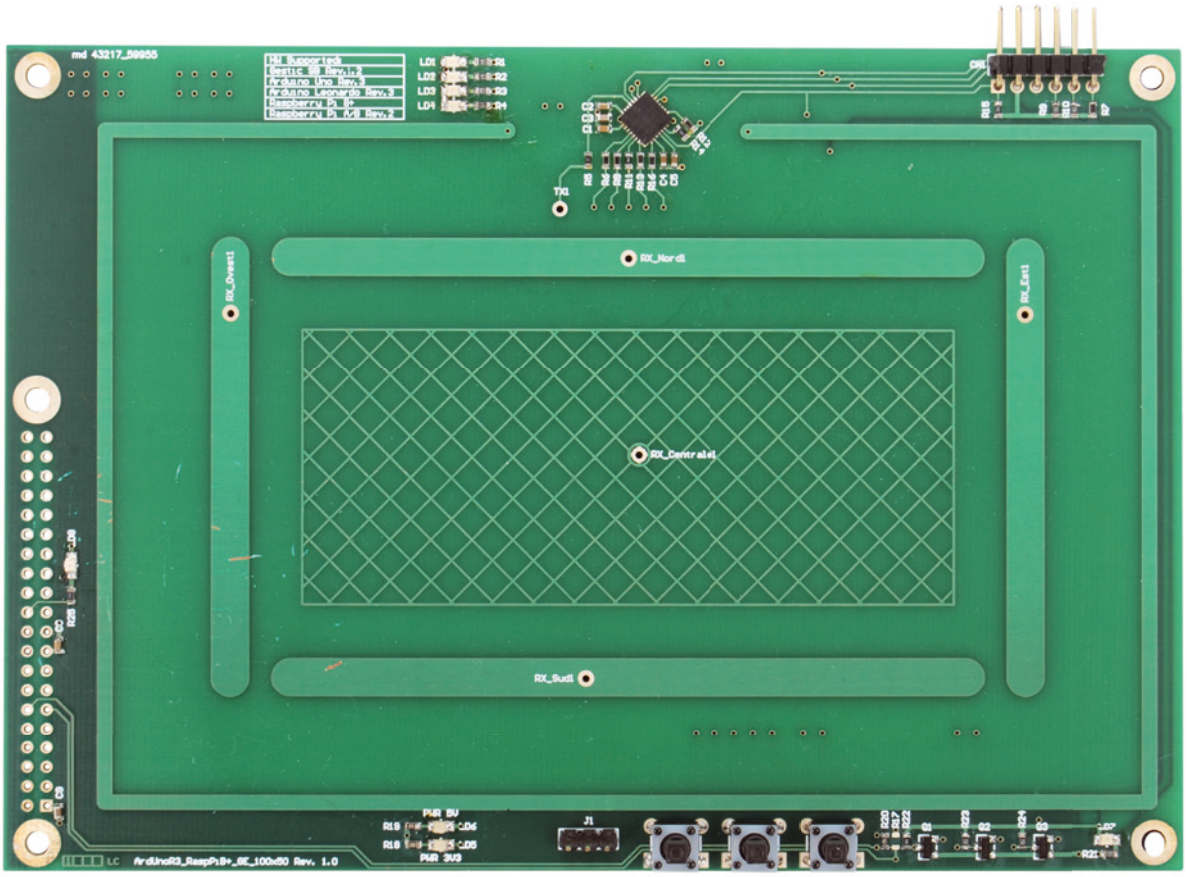


Elenco Componenti:

R1÷R4: 470 ohm 1% (0603)
R5: 0 ohm (0603)
R6÷R8: 10 kohm 1% (0603)
R9, R10: 1,8 kohm 1% (0603)
R11÷R16: 10 kohm 1% (0603)
R17: -
R18: 470 ohm 1% (0603)
R19, R21: 680 ohm 1% (0603)
R20: 5,2 kohm 1% (0603)
R22: 10 kohm 1% (0603)
R23, R24: 1,8 kohm 1% (0603)
R25: 680 ohm 1% (0603)
C1, C2: 100 nF ceramico (0603)
C3: 10 µF 6,3 VL ceramico (0603)
C4, C5: 4,7 µF 6,3 VL ceramico (0603)
C6÷C9: 10 µF 6,3 VL ceramico (0603)
Q1÷Q3: BSS123
U1: MGC3130-I/MQ

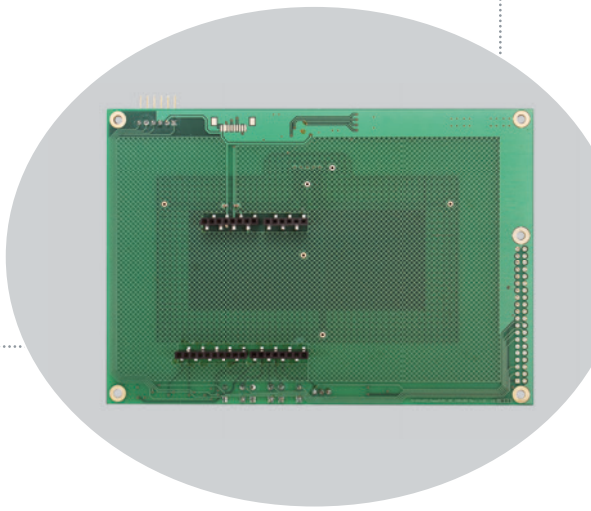
LIBRERIA ARDUINO MGC3130

In questo articolo ci concentreremo sull'accoppiata del nostro nuovo elettrodo con le schede Arduino Uno Rev.3 e Arduino Leonardo Rev.3.
Per entrambe abbiamo scritto due demo che si appoggiano sulla nostra libreria di gestione dell'integrato MGC3130.
La demo scritta per la scheda Arduino Uno Rev.3 viene completata dalla scheda di espansione FT1079K, sviluppata e venduta da Futura Elettronica (*www.futurashop.it*) la quale mette a disposizione 8 ingressi digitali e altrettante uscite a relé. Per la nostra demo useremo solo le otto uscite a relé, tuttavia, per chi lo desiderasse, è possibile gestire anche gli ingressi modificando opportunamente il codice della demo.
La gestione degli I/O viene fatta sfruttando l'integrato Microchip MCP23017, il quale viene connesso alle schede Arduino tramite il bus I²C come l'integrato MGC3130. Anche per l'MCP23017 abbiamo scritto una libreria di supporto che andremo a descrivere in breve durante l'articolo.
Grazie alle uscite a relé messe a disposizione, possiamo riportare le gesture riconosciute su una delle possibili uscite.
Per la nostra demo abbiamo



P1÷P3: Microswitch

Varie:
- Connettore 6 vie
- Connettore 8 vie (2 pz.)
- Connettore 10 vie
- Connettore WR-FPC 10 vie
- Pin strip femmina 2x20 vie
- Pin strip maschio 3 vie
- Jumper
- Pin strip maschio 6 vie 90°
- Circuito stampato S1214



deciso di riportare fino a un massimo di sedici gesture, per un totale di due schede FT1079K. Ovviamente è possibile aggiungere altre FT1079K per riportare più gesture possibili sulle uscite a relé (l'aggiunta di più di due schede FT1079K necessita di apportare modifiche allo sketch da noi scritto e può essere visto come un esercizio didattico interessante).
Invece la demo che riguarda la scheda Leonardo Rev.3 ci permette di interagire con il PC e in particolare con un programma di visualizzazione delle immagini. Grazie alle gesture riconosciute dall'integrato, sarà possibile sfogliare le immagini presenti sul PC.
Iniziamo a descrivere la libreria Arduino per la gestione dell'integrato MGC3130: essa è composta da un file con estensione .cpp (*MGC3130.cpp*) e da un file .h (*MGC3130.h*) contenente le dichiarazioni delle variabili e delle funzioni presenti nel file .cpp.
Oltre a questi c'è un file .txt (*keywords.txt*) con i riferimenti alla funzioni pubbliche utilizzate negli sketch Arduino. Le funzioni pubbliche messe a disposizione

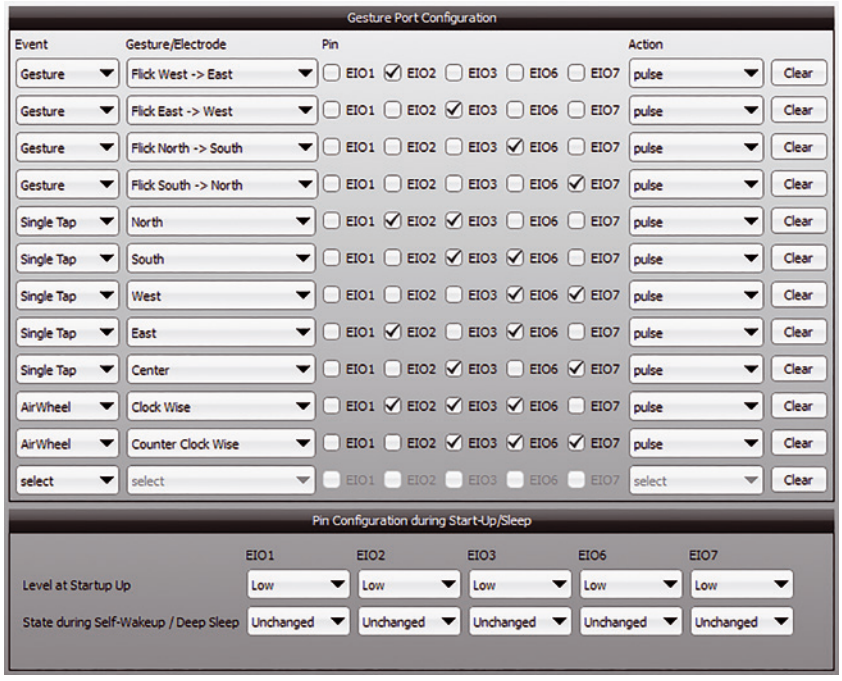


Fig. 4

dalla libreria sono le seguenti:

```
void SetSerial( uint8_t Baud, uint8_t Config)
void SetAdd(uint8_t Addr)
void ResetDevice(uint8_t Rst)
void ExitResetDevice(uint8_t Rst)
void Begin( uint8_t Ts, uint8_t Rst)
boolean GetTsLineStatus( uint8_t Ts)
void ReleaseTsLine( uint8_t Ts)
void GetEvent(void)
void DecodeGesture(void)
```

La funzione "***SetSerial***" serve semplicemente per inizializzare la comunicazione seriale da utilizzarsi con il "***monitor seriale***" presente nello IDE per eseguire il monitoraggio delle gesture riconosciute dall'integrato MGC3130. I parametri da passare alla funzione sono la velocità di comunicazione che si vuole utilizzare e la configurazione dei dati, ovvero lunghezza pacchetto dati, bit di parità bit, di stop ecc.
La funzione "***SetAdd***" serve per assegnare l'indirizzo hardware all'integrato MGC3130: nel nostro caso è sempre e solo 0x42.
La funzione "***ResetDevice***" serve per mantenere resettato l'integrato MGC3130, fintantoché non viene rimossa la condizione di reset. L'unico parametro da passare alla funzione è il pin cui è connessa la linea di RESET. La funzione opposta è "***ExitResetDevice***" la quale rimuove la condizione di RESET per l'integrato MGC3130; come per la precedente funzione, l'unico parametro è il pin a cui è collegata la linea di RESET.
La funzione "***Begin***" serve per inizializzare le linee di comunicazione tra la scheda Arduino e MGC3130, come parametri necessita soltanto i riferimenti per la linea TS e RESET. La funzione inizializza la periferica I²C con la consueta funzione "***WIRE.begin***" cui si deve passare l'indirizzo hardware dell'integrato MGC3130. La nostra libreria per gestire la comunicazione I²C sfrutta la libreria standard di sistema "***Wire***" includendo nel file .cpp il file di definizione "***Wire.h***".
Oltre all'inizializzazione dell'interfaccia I²C, si definiscono e configurano le linee RESET (Output) e TS (Input). La linea di RESET viene mantenuta a livello basso per un tempo di 250 ms, consentendo così la stabilizzazione dell'alimentazione al power-up.
Per indicare l'inizio e la fine della sequenza di inizializzazione, vengono stampati sul monitor seriale una serie di stringhe di testo.
Per scrivere del testo sul monitor seriale si utilizza la direttiva "***print***" o "***println***" presenti nella libreria "***Serial***". Tuttavia ogni qual volta si vuole scrivere una stringa di testo utilizzando queste direttive si spreca della memoria SRAM in quanto la stringa viene memorizzata proprio in questo tipo di memoria.
Per ovviare a questo inconveniente si può pensare di salvare le stringhe di testo nella memoria Flash, per poi andare a rileggerle e passarle alle funzione "***print***" come array di byte evitando di sprecare memoria SRAM. Quindi per memorizzare in Flash le stringhe di testo si può usare la seguente direttiva:

```
const char MGC3130Ready[] PROGMEM =
"MGC3130 device is ready";
```

Per rileggere la stringa memorizzata utilizziamo una funzione detta "***pgm_read_byte_near***" alla quale dobbiamo passare l'indirizzo di memoria Flash in cui è memorizzato il dato da leggere.
Abbiamo quindi scritto una nostra funzione che, tramite un loop, rilegge tutti i byte che compongono la stringa e la stampano sul monitor seriale.
Alla funzione bisogna quindi passare un puntatore in Flash che identifica l'inizio della stringa, la lunghezza della stringa e un boolean che indica alla funzione se alla fine deve stampare anche il carattere "\n".
Quindi, concludendo l'esempio della stringa di cui sopra, avremo:

```
ReadStringFLASH((uint8_t *)MGC3130Ready,
strlen(MGC3130Ready), TRUE);
```

La funzione "***GetTsLineStatus***" serve per monitorare la linea TS e rilevare quando l'integrato MGC3130 la impegna per avvisare che è stata riconosciuta una gesture. Non appena il sistema si accorge che la linea TS è stata impegnata, cambia lo stato del pin andando esso stesso ad occupare la linea TS, potendo così iniziare la procedura di lettura dei dati dall'integrato MGC3130. I dettagli sulla gestione della linea TS li trovate nell'articolo pubblicato nel fascicolo n° 195.
Conclusa la lettura dei dati, si può richiamare la funzione "***ReleaseTsLine***" rilasciando così la linea Ts.
Per entrambe le funzioni appena esposte, l'unico parametro che bisogna passargli è il pin a cui è collegata la linea TS. La funzione "***GetEvent***" serve per leggere i dati memorizzati nel buffer dell'integrato MGC3130 e salvarli in apposite strutture dati utilizzate dalla libreria; sarà poi una successiva funzione, quella che provvederà a decodificare i dati ricevuti e a renderli disponibili all'utente finale per la propria applicazione.
Vi ricordiamo che il pacchetto dati utilizzato per la lettura dei dati caricati nel buffer dell'integrato MGC3130 è del tipo mostrato in **Fig. 5**.
L'indirizzo contenuto nel pacchetto è quello hardware dell'integrato dove il bit meno significativo indica se si sta eseguendo una scrittura ("0") oppure una lettura ("1"). La sezione "***MGC3130 message***" può essere espansa come indicato in **Fig. 6** identificando due sezione "***Header***" e "***Payload***". Espandendo ancora la sezione "***Header***", come in **Fig. 7**, si nota che nei primi quattro byte ritroviamo la lunghezza del pacchetto dati ricevuto e, cosa molto importante, l'ID che identifica il pacchetto dati. Nel caso di lettura dei dati dopo il riconoscimento di una gesture, l'ID del pacchetto sarà 0x91. Per l'elenco dei codici ID disponibili vi invitiamo a rileggere l'articolo nel fascicolo n° 195.
Dei dati letti dalla funzione "***GetEvent***" durante il riconoscimento delle gesture, solo una parte viene tenuta in considerazione e in particolare le "***GestureInfo***" e le "***TouchInfo***". Queste vengono ulteriormente filtrate da delle costanti di mascheratura "***MASK_GESTURE_RAW***" e "***MASK_TOUCH_RAW***" eliminando le parti non necessarie. Per completezza vengono anche salvate le informazioni delle coordinate x, y e z.
Infine abbiamo la funzione "***DecodeGesture***" la quale decodifica i dati letti dalla precedente funzione e le rende disponibili attraverso una struttura dati pubblica che l'utente può utilizzare nella propria applicazione. Le gesture riconosciute e messe a disposizione dalla libreria sono:

- ***Gesture Touch South***;
- ***Gesture Touch West***;
- ***Gesture Touch North***;
- ***Gesture Touch East***;
- ***Gesture Touch Centre***;
- ***Gesture Tap South***;
- ***Gesture Tap West***;
- ***Gesture Tap North***;
- ***Gesture Tap East***;
- ***Gesture Tap Centre***;
- ***Gesture Double Tap South***;
- ***Gesture Double Tap West***;
- ***Gesture Double Tap North***;
- ***Gesture Double Tap East***;
- ***Gesture Double Tap Centre***;
- ***Flick West to East***;
- ***Flick East to West***;
- ***Flick South to North***;
- ***Flick North to South***;
- ***Edge Flick West to East***;
- ***Edge Flick East to West***;
- ***Edge Flick South to North***;
- ***Edge Flick North to South***;
- ***Clock Wise***;
- ***Counter Clock Wise***.

La libreria si accorge delle variazioni intercettate dall'integrato MGC3130 e le riporta sui rispettivi bit della struttura dati pubblica. La gesture è riconosciuta quando il bit corrispondente va a "1" logico. La

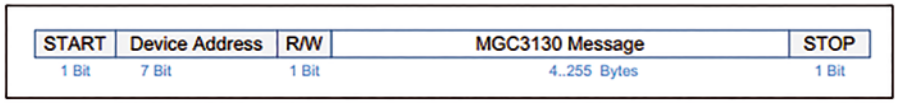


Fig. 5

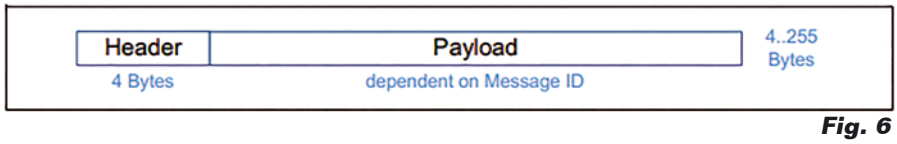


Fig. 6

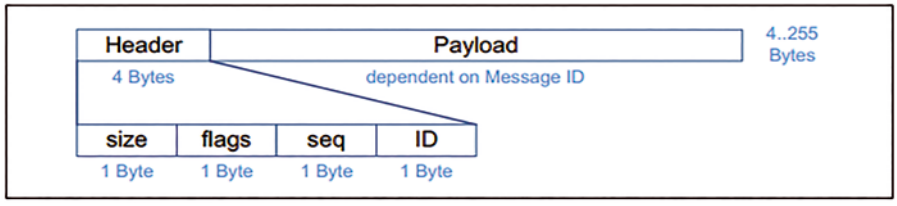


Fig. 7

Listato 1

```
union GestureOutput {
        uint32_t Gesture;
        uint8_t  GestArray[4];
        struct {
                uint8_t  Byte_0;
                uint8_t  Byte_1;
                uint8_t  Byte_2;
                uint8_t  Byte_3;
        } GestureByte;
        struct {
                uint8_t TouchSouth              :1;  // GESTURE_TOUCH_SOUTH              0x00000001
                uint8_t TouchWest               :1;  // GESTURE_TOUCH_WEST               0x00000002
                uint8_t TouchNorth              :1;  // GESTURE_TOUCH_NORTH              0x00000004
                uint8_t TouchEast               :1;  // GESTURE_TOUCH_EAST               0x00000008
                uint8_t TouchCentre             :1;  // GESTURE_TOUCH_CENTRE             0x00000010
                uint8_t TapSouth                :1;  // GESTURE_TAP_SOUTH                0x00000020
                uint8_t TapWest                 :1;  // GESTURE_TAP_WEST                 0x00000040
                uint8_t TapNorth                :1;  // GESTURE_TAP_NORTH                0x00000080
                uint8_t TapEast                 :1;  // GESTURE_TAP_EAST                 0x00000100
                uint8_t TapCentre               :1;  // GESTURE_TAP_CENTRE               0x00000200
                uint8_t DoubleTapSouth          :1;  // GESTURE_DOUBLE_TAP_SOUTH         0x00000400
                uint8_t DoubleTapWest           :1;  // GESTURE_DOUBLE_TAP_WEST          0x00000800
                uint8_t DoubleTapNorth          :1;  // GESTURE_DOUBLE_TAP_NORTH         0x00001000
                uint8_t DoubleTapEast           :1;  // GESTURE_DOUBLE_TAP_EAST          0x00002000
                uint8_t DoubleTapCentre         :1;  // GESTURE_DOUBLE_TAP_CENTRE        0x00004000
                uint8_t GestWestEast            :1;  // GESTURE_WEST_EAST                0x00008000
                uint8_t GestEastWest            :1;  // GESTURE_EAST_WEST                0x00010000
                uint8_t GestSouthNorth          :1;  // GESTURE_SOUTH_NORTH              0x00020000
                uint8_t GestNorthSouth          :1;  // GESTURE_NORTH_SOUTH              0x00040000
                uint8_t EdgeGestWestEast        :1;  // GESTURE_EDGE_WEST_EAST           0x00080000
                uint8_t EdgeGestEastWest        :1;  // GESTURE_EDGE_EAST_WEST           0x00100000
                uint8_t EdgeGestSouthNorth      :1;  // GESTURE_EDGE_SOUTH_NORTH         0x00200000
                uint8_t EdgeGestNorthSouth      :1;  // GESTURE_EDGE_NORTH_SOUTH  0      x00400000
                uint8_t GestClockWise           :1;  // GESTURE_CLOCK_WISE               0x00800000
                uint8_t GestCounterClockWise    :1;  // GESTURE_COUNTER_CLOCK_WISE       0x01000000
                uint8_t FreeBit                 :7;
        } Bit;
} GestureOutput;
```

struttura dati pubblica è distribuita su 32 bit, di cui 7 liberi per sviluppi futuri. Inoltre è possibile leggere e modificare i dati della struttura agendo su ogni singolo bit oppure più brutalmente a livello di byte o double word.
Nel **Listato 1** riportiamo la struttura dati nella sua interezza. Quindi per testare un singolo bit, ad esempio "***TapWest***", si dovrà scrivere la seguente:

```
if (mgc3130.GestureOutput.Bit.TapWest != 0) {… … …
```

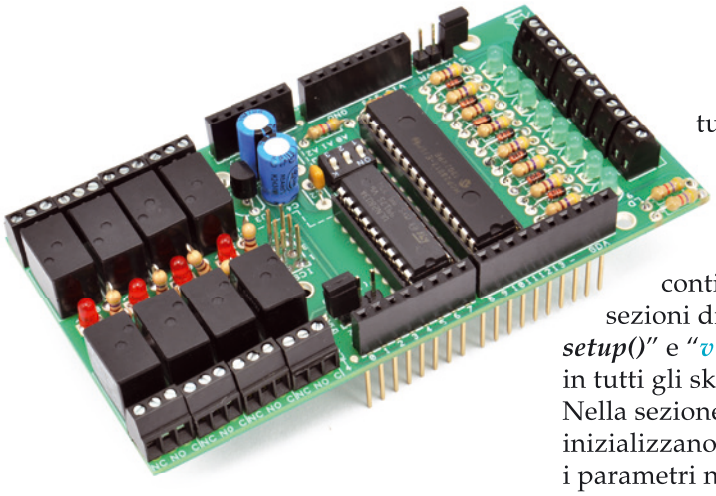
dove "**mgc3130** "è il riferimento alla libreria incluso nello sketch. Volendo, ma non è necessario, è possibile filtrare le gesture che non si vogliono utilizzare nel proprio progetto. Infatti è possibile configurare una costante di filtro, presente nel file MGC3130.h, che permette di filtrare le gesture non volute. La costante si chiama "***MASK_FILTER_GESTURE***" ed è un valore a 32 bit configurabile bit a bit. Se il bit è a "***1***" logico significa che si desidera filtrare la gesture viceversa la si vuole mantenere. Nei nostri sketch di esempio non abbiamo fatto uso della costante di filtro, in altre parole l'abbiamo lasciata configurata a 0x00000000, questo vuol dire che nella funzione "***DecodeGesture***" è usata ma non ha alcuna influenza.
Per favorire lo studio della libreria, ci sono delle parti di codice nascoste che possono essere attivate al bisogno; queste sezioni sono racchiuse tra le direttive "***#ifdef identifier***" e "***endif***" dove ***identifier*** è una etichetta che, quando dichiarata, attiva la sezione di codice viceversa la esclude.
Ad esempio la porzione di codice seguente:

```
#ifdef PRINT_RAW_DATA
        PrintMGC3130RawData();
#endif
```

attiva/disattiva il codice della funzione "***PrintMGC3130 RawData***". Tale funzione stampa sul monitor seriale i dati letti dall'integrato prima che vengano decodificati dalla funzione "***DecodeGesture***". Durante lo studio della libreria, e dell'integrato MGC3130, può risultare molto utile perché mostra i dati esattamente come escono dal buffer di MGC3130. Ad esempio la gesture "***Flick East to West***" è così evidenziata:

```
##############################
Row data from MGC3130
Header: 1A081191
Payload: 1F01 | 86 | 80 | 0073 | 03100000
| 00000000 | 0000 | 000000000000
##############################
```

Per aumentarne la leggibilità



e interpretazione il pacchetto dati viene diviso tra "***Header***" e "**Payload**", in più il "***Payload***" viene suddiviso in sotto-pacchetti come evidenziato nel data-sheet della libreria Gestic.
Oltre alla sezione di codice nascosta appena trattata, ce ne sono altre tre, tra cui le gesture riconosciute e decodificate e le coordinate X, Y e Z (rispettivamente ***PRINT_GESTURE_DATA*** e ***PRINT_XYZ***).
Se non attivate queste funzioni, esse non occupano spazio nella Flash.

MGC3130_DEMO

Descriviamo ora il nostro primo sketch, nel quale mostriamo l'utilizzo della nostra libreria e l'interfacciamento delle gesture selezionate rispetto alle uscite a relé messe a disposizione delle schede FT1079K.
Lo sketch è suddiviso in quattro file, di cui quello principale si chiama "***MGC3130_Demo***".
Gli altri file permettono la gestione degli ingressi digitali messi a disposizione dal nostro elettrodo ("***DigitalInput***"), la gestione delle uscite a relé e del riporto delle gesture sulle uscite ("***DigitalOutput***"), e un ultimo file per la gestione del TIMER1 del microcontrollore Atmel ("***TimersInt***") utile per gestire tutte le costanti di tempo utilizzate nello sketch.
Il file "***MGC3130_Demo***" contiene le due classiche sezioni di codice "***void setup()***" e "***void loop()***" presenti in tutti gli sketch Arduino.
Nella sezione "***void setup()***" si inizializzano e configurano tutti i parametri necessari al corretto funzionamento dello sketch, nonché le schede di espansione FT1079K con indirizzo hardware 0x00 e 0x01, nonché l'integrato MGC3130 (Indirizzo 0x42).
Vengono inizializzati i pin di ingresso, richiamando la funzione "***void SetInputPin(void)***", necessari a gestire i tre pulsanti P1, P2 e P3 e il jumper J1.
Vengono configurati i pin di uscita collegati alla scheda Arduino, in questo caso il solo led LD7 e infine viene configurato il TIMER1 e il relativo vettore di interrupt per la gestione delle costanti di tempo.
Per ultimo, ma non meno importanti, vengono configurate le macchine a stati utilizzate nello sketch.
La sezione "***void loop()***" presenta i richiami a tutte le macchine a stati precedentemente inizializzate nonché il codice per gestire l'integrato MGC3130. In particolare il codice di nostro interesse è il seguente:

```
if (mgc3130.GetTsLineStatus(Ts_MGC3130) == 0) {
   mgc3130.GetEvent();     //  Start read data from MGC3130
   mgc3130.DecodeGesture();   //  Decode Gesture
   mgc3130.ReleaseTsLine(Ts_MGC3130);   //  Release TS Line
}
```

Utilizzando la funzione di libreria:

```
boolean GetTsLineStatus( uint8_t Ts)
```

si testa la linea TS in attesa che vada a livello logico basso per poi impegnarla e iniziare la procedura di lettura dei dati presenti nel buffer di MGC3130 tramite la funzione di libreria:

```
void GetEvent(void)
```

Viene poi richiamata la funzione per la decodifica dei dati letti per estrapolare le gesture riconosciute e infine si rilascia la linea TS:

```
void DecodeGesture(void)
void ReleaseTsLine(uint8_t Ts)
```

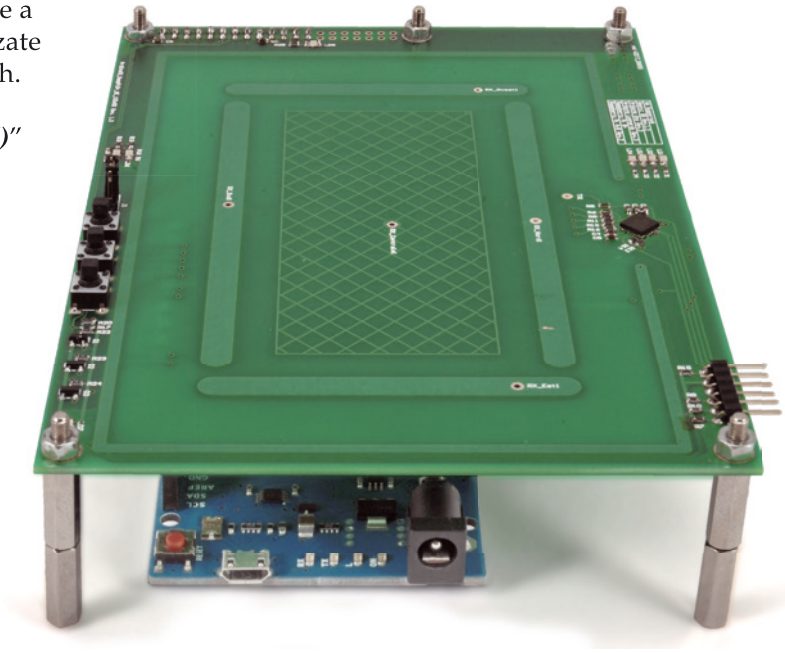


Tabella 1

P1	P2	P3	Tempo selezionato
ON	OFF	OFF	Tempo monostabile 1 secondo. Premere pulsante P1 per più di due secondi
OFF	ON	OFF	Tempo monostabile 5 secondi. Premere pulsante P2 per più di due secondi
OFF	OFF	ON	Tempo monostabile 10 secondi. Premere pulsante P3 per più di due secondi
ON	ON	OFF	Tempo monostabile 15 secondi. Premere pulsanti P1 & P2 per più di due secondi
OFF	ON	ON	Tempo monostabile 20 secondi. Premere pulsanti P2 & P3 per più di due secondi
ON	OFF	ON	Tempo monostabile 25 secondi. Premere pulsanti P1 & P3 per più di due secondi
ON	ON	ON	Tempo monostabile 30 secondi. Premere pulsanti P1 & P2 & P3 per più di due secondi

Tabella 2

Uscita	Indirizzo FT1079K	Gesture
01	0x00	Tap su elettrodo Sud
02	0x00	Tap su elettrodo Ovest
03	0x00	Tap su elettrodo Nord
04	0x00	Tap su elettrodo Est
05	0x00	Tap elettrodo Centrale
06	0x00	Doppio Tap su elettrodo Sud
07	0x00	Doppio Tap su elettrodo Ovest
08	0x00	Doppio Tap su elettrodo Nord
01	0x01	Doppio Tap su elettrodo Est
02	0x01	Doppio Tap elettrodo Centrale
03	0x01	Gesture movimento Ovest verso Est
04	0x01	Gesture movimento Est verso Ovest
05	0x01	Gesture movimento Sud verso Nord
06	0x01	Gesture movimento Nord verso Sud
07	0x01	Gesture movimento circolare orario
08	0x01	Gesture movimento circolare antiorario

Il jumper J1 collegato all'ingresso 3 (PD3) serve per decidere se le uscite digitali si devono comportare come monostabili o bistabili. Se il jumper è inserito, e quindi l'ingresso è a livello logico basso, le uscite devono funzionare in modo monostabile, viceversa come bistabili. Nella modalità monostabile è possibile decidere il tempo di eccitazione dell'uscita sfruttando i tre pulsanti P1, P2 e P3 (pulsanti collegati rispettivamente agli ingressi 4, 5, e 6 ovvero PD4, PD5 e PD6).
Tenendo premuto il pulsante P1 per più di due secondi, verrà selezionato come tempo per il monostabile 1 secondo, se viene premuto P2 per più di due secondi verrà selezionato un tempo di 5 secondi ecc. ecc. Riportiamo nella **Tabella 1** le possibili combinazioni con i pulsanti P1, P2 e P3 e i rispettivi tempi selezionabili (ON → Premuto; OFF → Non Premuto). Il tempo che viene selezionato è globale e quindi viene associato a tutte le uscite digitali pilotate come monostabili. In altre parole una volta selezionato, ad esempio, un tempo di 5 secondi questo sarà associato a tutte le uscite monostabili. Non è possibile associare tempi diversi per ogni uscita.
Notate che la condizione bistabile significa che l'uscita inverte la propria condizione ad ogni riconoscimento della gesture associata; quindi un rilevamento della gesture eccita l'uscita, un successivo rilevamento la porta a riposo e via di seguito. La modalità monostabile, invece, eccita l'uscita associata alla gesture e la mantiene per il tempo selezionato (vedere **Tabella 1**). Se viene intercettata nuovamente la gesture associata all'uscita prima che il tempo sia scaduto, si avrà un riarmo della stessa prolungando l'eccitamento dell'uscita associata.
Come accennato durante la descrizione della libreria, le gesture riconosciute e messe a disposizione dell'utente sono in totale 25; di queste, solo una parte è riportata sulle uscite a relé gestire dalle schede FT1079K. Nella **Tabella 2** riportiamo la corrispondenza tra la gesture riconosciuta e l'uscita a relé indirizzata sulla scheda FT1079K. Volendo, è possibile riportare sulle uscite a relé anche le gesture mancanti, ma per fare ciò è necessario aggiungere almeno un'altra scheda FT1079K con indirizzo 0x02; fatto ciò, bisognerà chiaramente scrivere il codice necessario alla gestione della nuova scheda.

MGC3130_LEONARDO

Descriviamo ora la demo realizzata con la scheda Arduino Leonardo Rev. 3. Lo scopo di questo sketch è di comandare un software di gestione delle immagini attraverso le gesture riconosciute dall'integrato MGC3130. Prima di tutto parliamo del programma utilizzato per la demo in esame: si tratta del programma "***FastStone Image Viewer***" scaricabile gratuitamente dal link web *www.faststone.org/FSViewerDownload.htm*.
Dopo avere installato il software sul vostro PC, procedete con il configurare il suo possibile avvio tramite una combinazione di

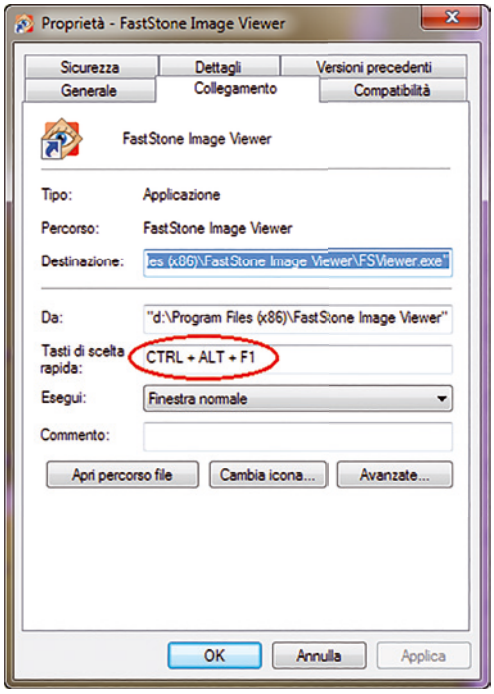


Fig. 8

tasti; per fare ciò bisogna cliccare con il pulsante destro del mouse sull'icona del programma e sotto la voce "***Collegamento***" impostare i tasti di scelta rapida. Nel nostro caso abbiamo impostato la combinazione "***CTRL+ALT+F1***" come evidenziato dalla **Fig. 8.**
Con questo accorgimento possiamo impostare una gesture per l'avvio del software e altrettante gesture per la gestione delle immagini. Il meccanismo di interazione con il software su PC tramite gesture deve essere attivato; per fare ciò abbiamo predisposto due possibili azioni: la prima consiste nell'utilizzare i pulsanti P1 e P2 presenti sulla scheda elettrodo. Il pulsante P1, se premuto per più di due secondi, attiva il meccanismo di gestione del software su PC; se invece si preme il pulsante P2 per più di due secondi, il meccanismo viene disattivato. L'altro metodo consiste nell'utilizzare il monitor seriale, per attivare il meccanismo inviare il carattere ASCII '***S***' viceversa '***P***' per disattivarlo. L'attivazione tramite pulsante P1 o carattere ASCII '***S***' richiama le funzioni di attivazione dei servizi keyboard e mouse messi a disposizione dalla scheda Leonardo:

```
Keyboard.begin();
Mouse.begin();
```

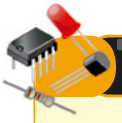
Viceversa il pulsante P2 o il carattere ASCII '***P***' richiama le funzioni di disattivazione dei servizi appena menzionati:

```
Keyboard.end();
Mouse.end();
```

L'attivazione/disattivazione tramite caratteri ASCII viene utile nel caso in cui la scheda Arduino Leonardo Rev.3 venga collegata a delle schede elettrodo diverse da quelle utilizzate per lo sketchup. Ad esempio gli elettrodi presentati nelle precedenti puntate o altro. Eseguita questa manovra preliminare, il sistema è pronto per riconoscere le gesture e di conseguenza di interagire con il software di gestione delle immagini.
Per aprire il programma è sufficiente un Tap sull'elettrodo centrale: il programma viene aperto e saranno mostrate le immagini a disposizione. Per sfogliare le immagini utilizzare le gesture "***Ovest verso Est***" e "***Est verso Ovest***": la prima sfoglia le immagini verso destra e la seconda verso sinistra. Per vedere le immagini a tutto schermo è sufficiente eseguire un Tap sull'elettrodo "***Ovest***" e per uscire dalla visualizzazione a schermo intero è sufficiente un Tap sull'elettrodo "***Est***".
Per ingrandire una immagine (Zoom +) eseguire una gesture "***Edge Ovest verso Est***" e per ridurre l'immagine (Zoom -) eseguire una gesture "***Edge Est verso Ovest***". Quando si guarda una immagine a schermo intero zoomata è possibile muovere l'immagine ingrandita con le gesture "***Ovest verso Est***", "***Est verso Ovest***", "***Nord verso Sud***" e "***Sud verso Nord***".
Infine è possibile ruotare le immagini selezionate, per fare ciò eseguire una movimento circolare in senso orario per ruotare l'immagine verso destra e un movimento in senso antiorario per ruotare l'immagine verso sinistra. Per chiudere il software è sufficiente un doppio Tap sull'elettrodo centrale.

CONCLUSIONI

In questa terza puntata dedicata alla tecnologia GestIC abbiamo imparato a utilizzare il nuovo elettrodo con Arduino Uno Rev. 3 e Arduino Leonardo Rev. 3, vi abbiamo dato una serie di spunti ed esempi per l'utilizzo delle nostre librerie, fornendovi i mezzi per implementare nuove e interessanti funzioni da utilizzare nei vostri progetti. Nel prossimo articolo vedremo l'applicazione con Raspberry Pi.



per il MATERIALE

Il software Aurea di Microchip può essere scaricato dal sito www.microchip.com/gestic.
Il sensore di prossimità (cod. MGC3130) è reperibile presso Futura Elettronica a 9,80 Euro.

Il materiale va richiesto a:
Futura Elettronica, Via Adige 11,
21013 Gallarate (VA)
Tel: 0331-799775 • Fax: 0331-792287
http://www.futurashop.it